ClassificationUsingNaiveBayes

October 20, 2025

[1]:
```python
# Pandas is for data manipulation and analysis (often used for DataFrames).
import pandas as pd

# load_iris loads the classic Iris dataset (machine learning demo dataset)␣
↪directly from scikit-learn.
from sklearn.datasets import load_iris

# train_test_split splits the dataset into training and testing sets easily.
from sklearn.model_selection import train_test_split

# GaussianNB is the Naïve Bayes classifier for continuous values (here: Iris␣
↪features).
from sklearn.naive_bayes import GaussianNB

# confusion_matrix creates a matrix comparing predictions vs actuals for␣
↪evaluating classifier errors.
from sklearn.metrics import confusion_matrix

# accuracy_score computes the percentage of correct predictions.
from sklearn.metrics import accuracy_score

# precision_score measures what fraction of predicted positives are correct␣
↪(used for classifier quality).
from sklearn.metrics import precision_score

# recall_score measures what fraction of actual positives were found by the␣
↪classifier (another quality metric).
from sklearn.metrics import recall_score

# classification_report gives a full summary (precision, recall, f1, etc.) for␣
↪ALL classes.
from sklearn.metrics import classification_report
```

[2]:
```python
# Loads the built-in Iris dataset from scikit-learn. Returns a 'Bunch' object,␣
↪similar to a dictionary, containing data, labels, and metadata.
iris = load_iris()
```

```
# Extracts the feature data (inputs) from the 'iris' object.
# X is a 2D numpy array of shape (150, 4) where each row is a flower sample and␣
↪each column is a feature (sepal/petal length/width).
X = iris.data

# Extracts the target values (outputs/classes) from the 'iris' object.
# y is a 1D numpy array of shape (150,) where each entry is a class label for␣
↪the corresponding sample in X.
y = iris.target
```

[3]:
```
# Splits the dataset into training and testing parts.
# X and y are split into four sets:
# - X_train: feature data for training (80% of total data)
# - X_test: feature data for testing (20% of total data)
# - y_train: target labels for training (80% of total labels)
# - y_test: target labels for testing (20% of total labels)
# test_size=0.2 means 20% of the data will be used for testing, and 80% for␣
↪training.
# random_state=42 ensures the split is reproducible; the same data will be␣
↪selected each time you run this code.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[4]:
```
# Creates an instance of the Gaussian Naïve Bayes classifier.
# This classifier assumes that features follow a normal (Gaussian) distribution.
model = GaussianNB()

# Trains (fits) the classifier on the training data.
# Uses X_train (input features) and y_train (target labels) to learn the␣
↪relationship between features and classes.
# After this step, the model can predict the class of new, unseen samples.
model.fit(X_train, y_train)
```

[4]:
```
GaussianNB()
```

[5]:
```
# Uses the trained model to predict the labels of the test feature data.
# y_pred will contain the predicted class labels for each sample in X_test.
y_pred = model.predict(X_test)
```

[6]:
```
# Confusion Matrix: Compares actual labels (y_test) to predicted labels␣
↪(y_pred).
# Each cell [i, j] tells you how often class i samples were predicted as class␣
↪j.
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

[7]:
```python
# Accuracy: Overall proportion of correct predictions.
# Formula: (number of correct predictions) / (total number of predictions)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
```

```
Accuracy: 1.0
```

[8]:
```python
# Precision: Fraction of predicted positives for each class that were correct␣
↪(useful in imbalanced classes).
# 'macro' means compute precision for each class, then take the average (treat␣
↪all classes equally).
precision = precision_score(y_test, y_pred, average='macro')
print("Precision:", precision)
```

```
Precision: 1.0
```

[9]:
```python
# Recall: Fraction of actual positives for each class that were correctly␣
↪predicted.
# 'macro' means compute recall for each class, then average-all classes matter␣
↪equally.
recall = recall_score(y_test, y_pred, average='macro')
print("Recall:", recall)
```

```
Recall: 1.0
```

[10]:
```python
# Classification Report: Gives precision, recall, f1-score, and support for␣
↪every class.
# target_names uses the original species names instead of numbers.
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[ ]:
```python
!jupyter nbconvert --to pdf "ClassificationUsingNaiveBayes.ipynb" --output "C:/
↪Users/ASUS/Downloads/Classification_NaiveBayes.pdf"
```

[ ]:
```python

```